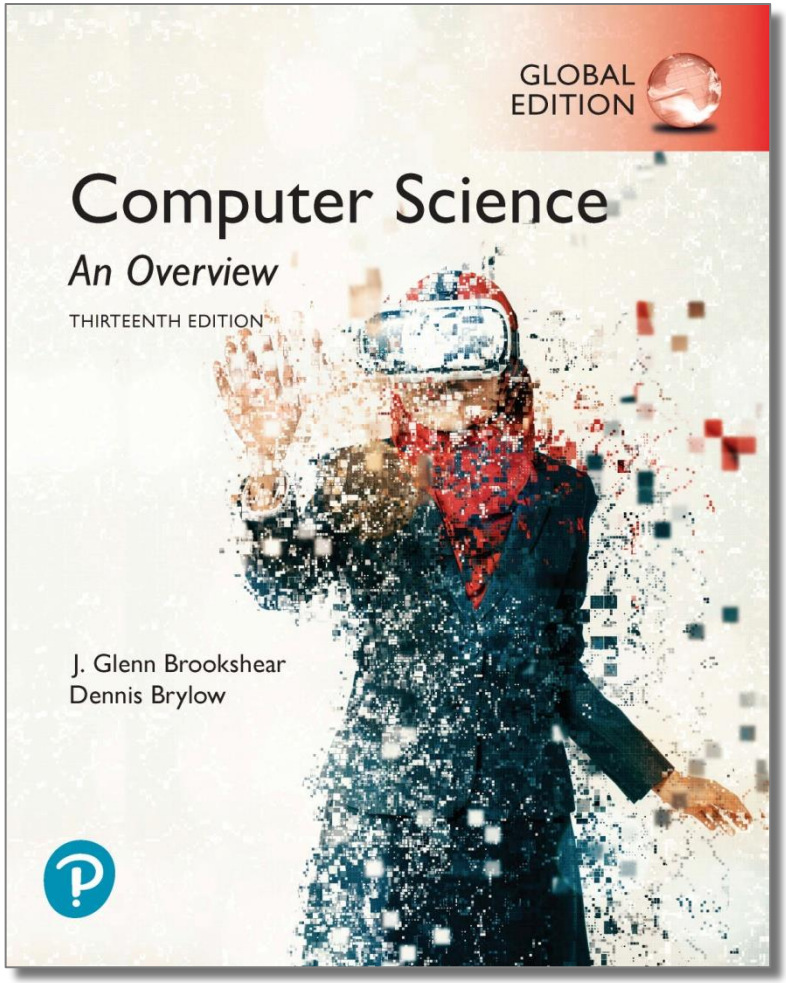
Computer Science An Overview

13th Edition, Global Edition



Chapter 2

Data Manipulation

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Chapter 2: Data Manipulation

- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution
- 2.4 Arithmetic/Logic
- 2.5 Communicating with Other Devices
- 2.6 Program Data Manipulation
- 2.7 Other Architectures

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

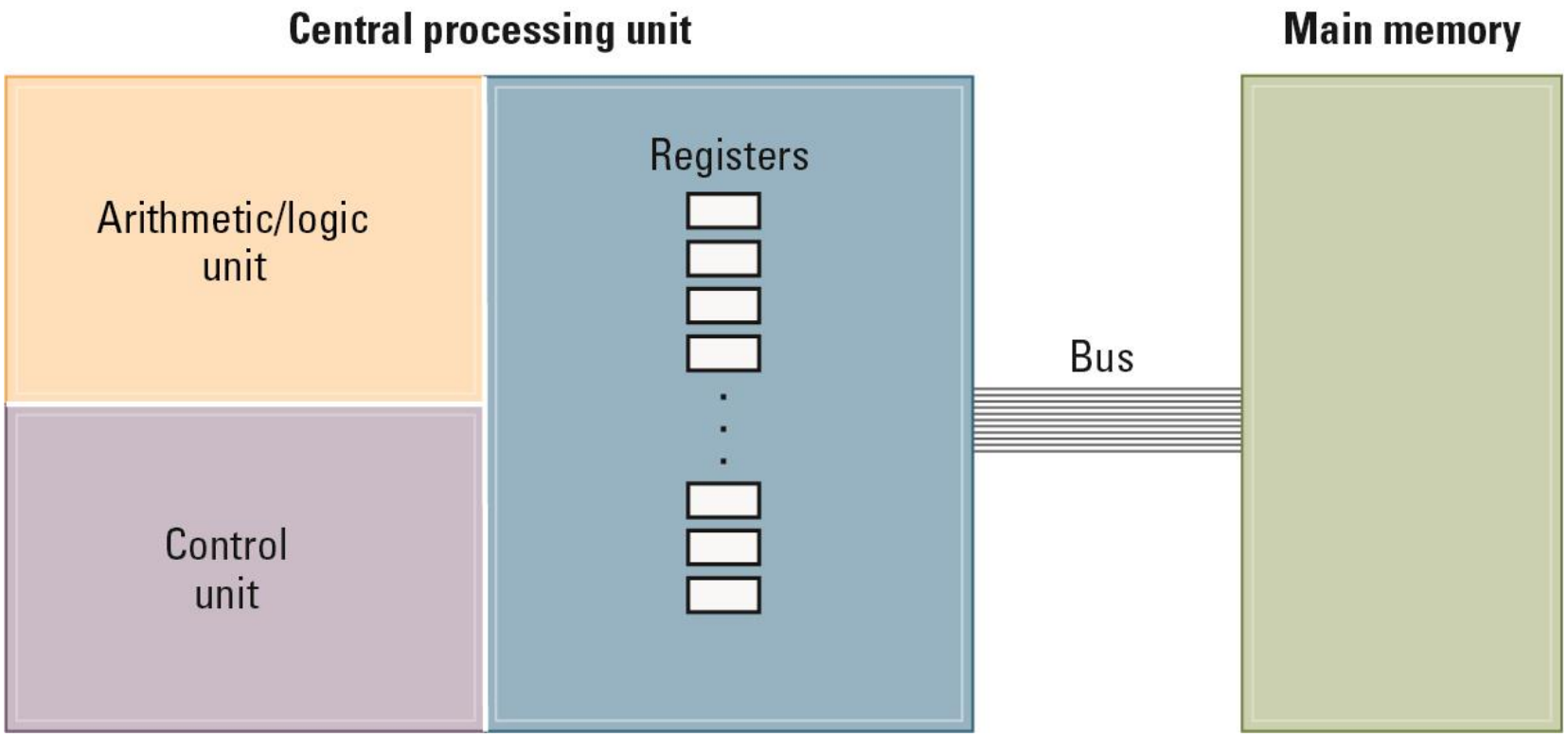
2.1 Computer Architecture

- Central Processing Unit (CPU)
  - Arithmetic/Logic Unit
  - Control Unit
  - Register Unit
    - General purpose registers
      - Store either data or address.
    - Special purpose registers
      - Stores data inside the CPU, such as program counters, stack registers, and state registers.
- Bus
- Main Memory

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.1 CPU and main memory connected via a bus

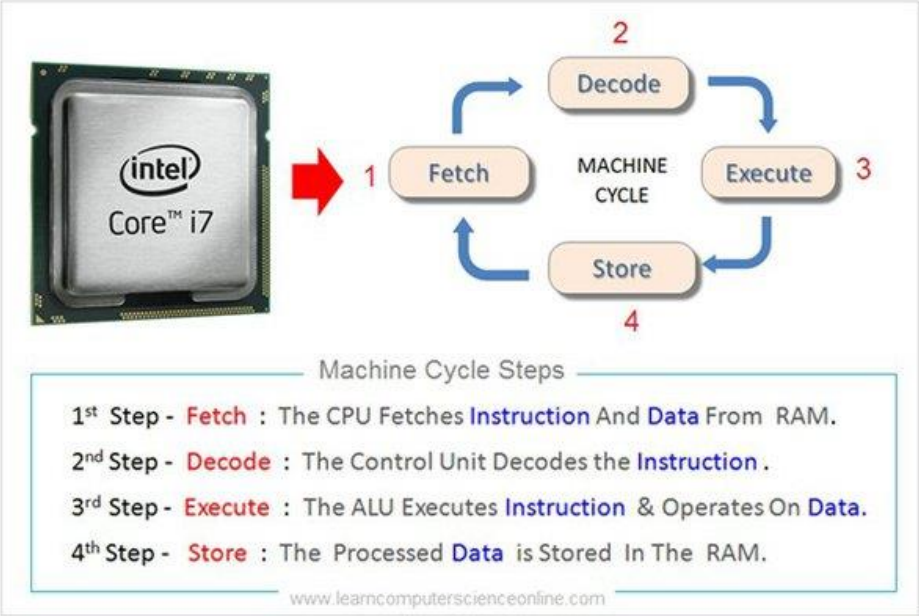


Pearson
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Stored Program Concept

A program can be encoded as bit patterns and stored in Main Memory. From there, the Control Unit can extract, decode, and execute instructions.

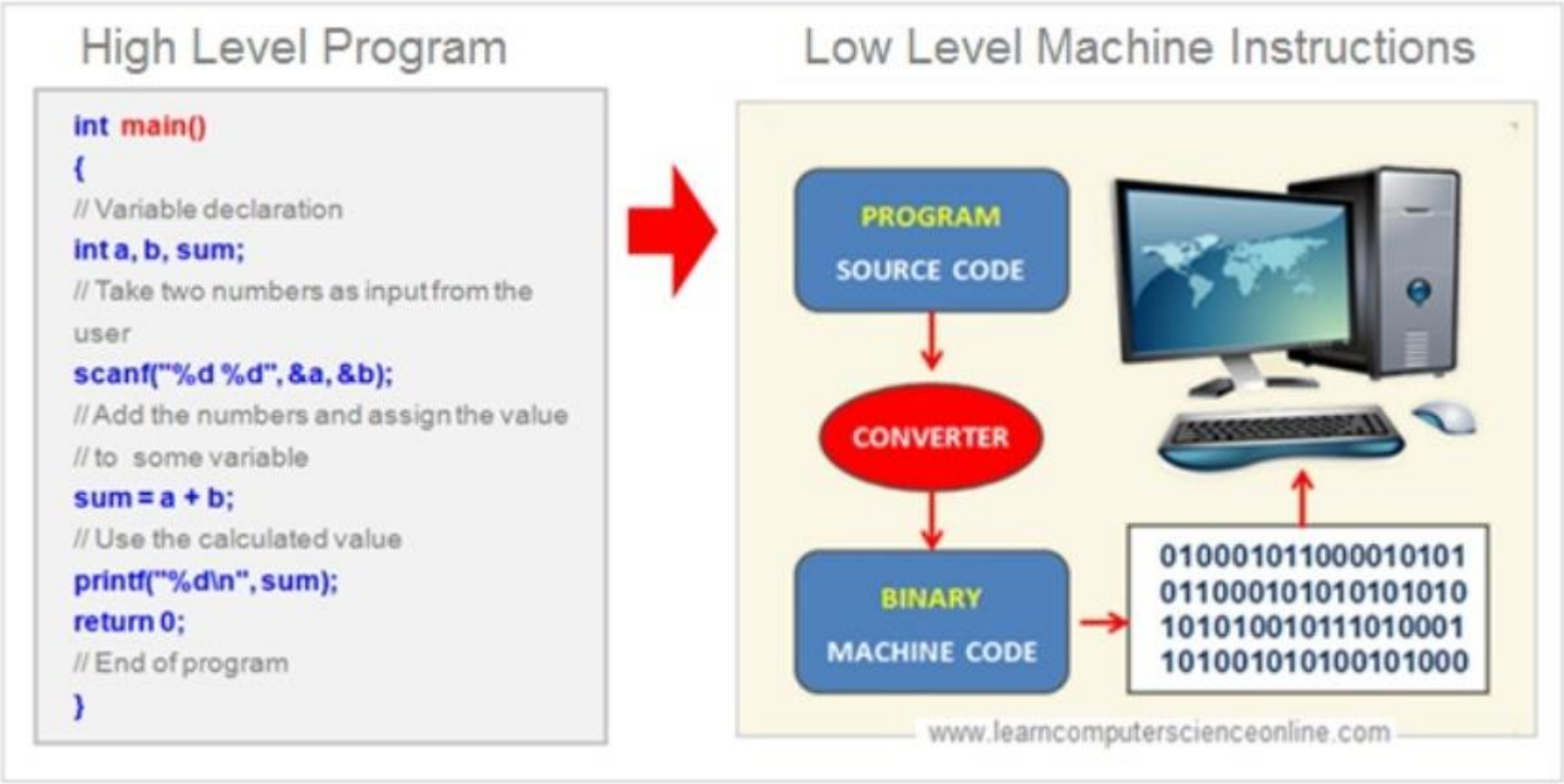
Instead of rewiring the CPU, the program can be altered by changing the contents of Main Memory.



Pearson

tion, Ltd. All Rights Reserved.

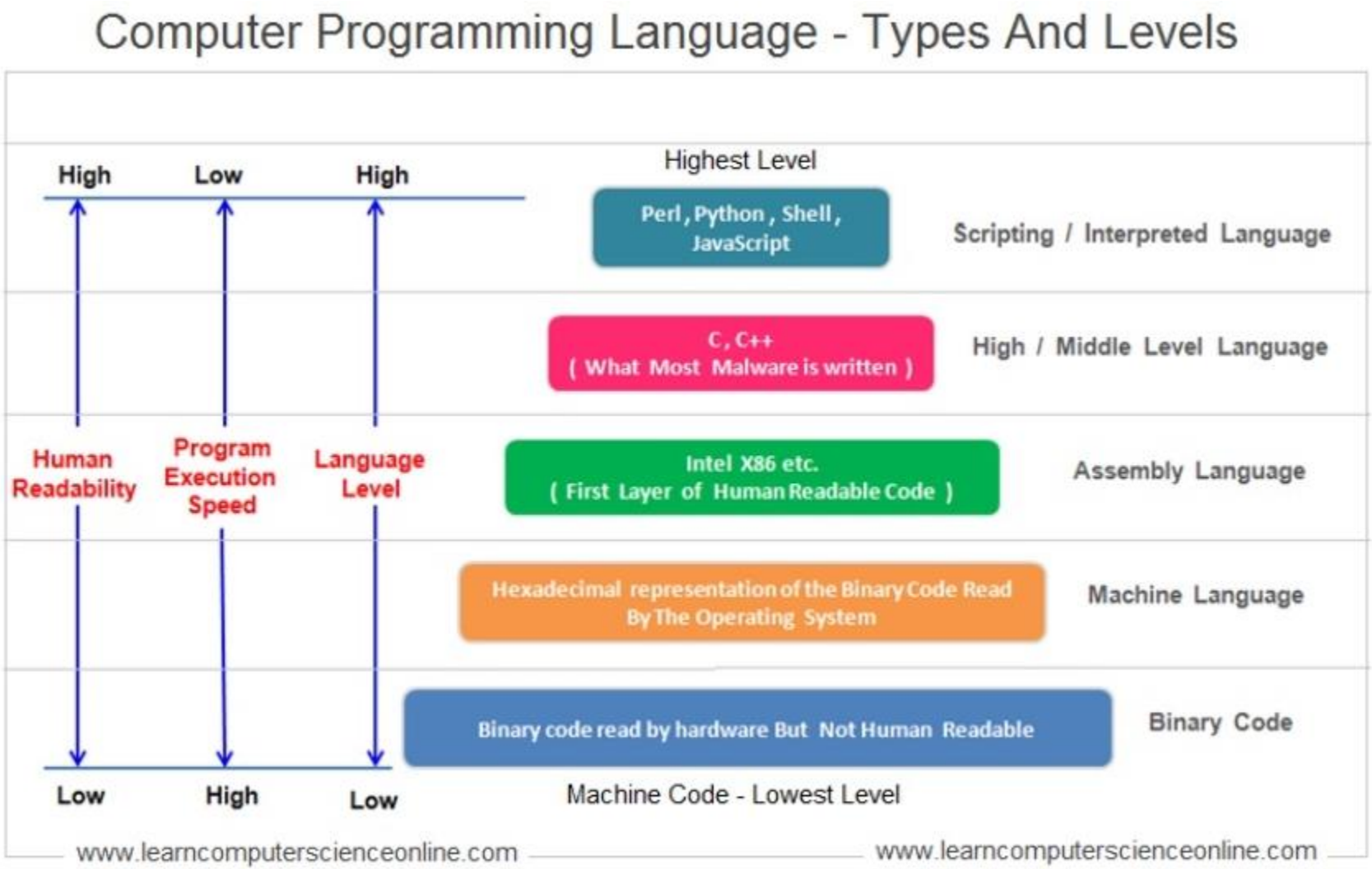
2.2 Machine Language



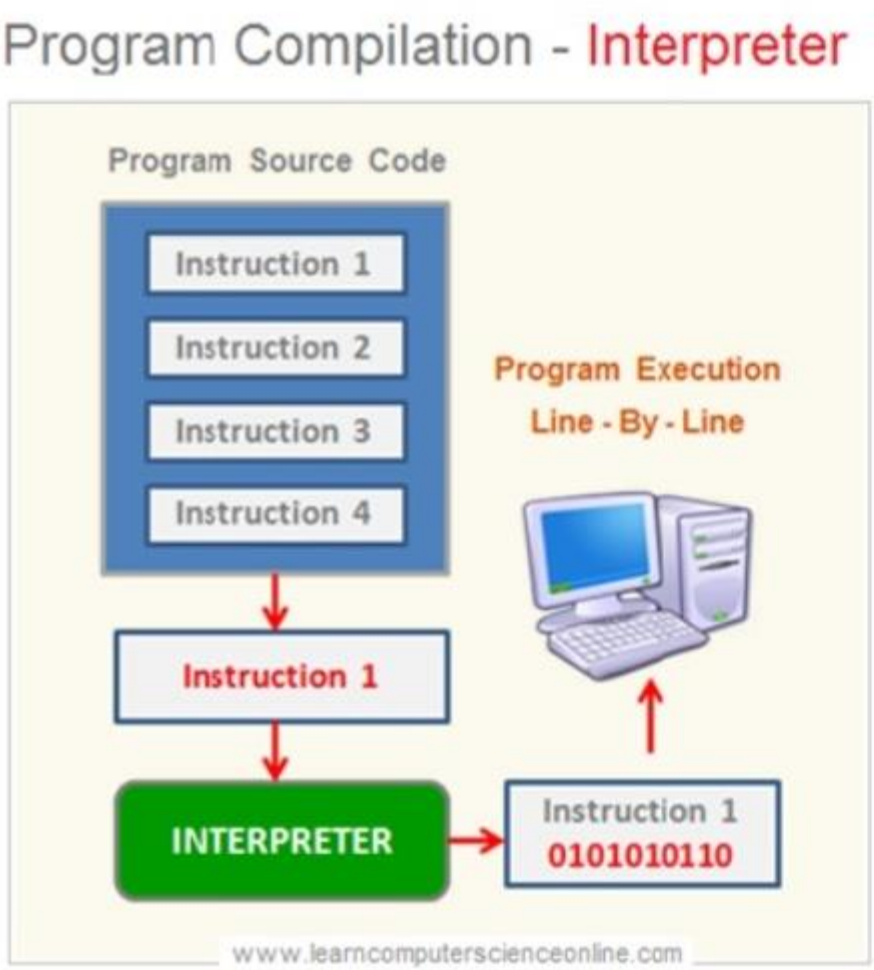
Pearson

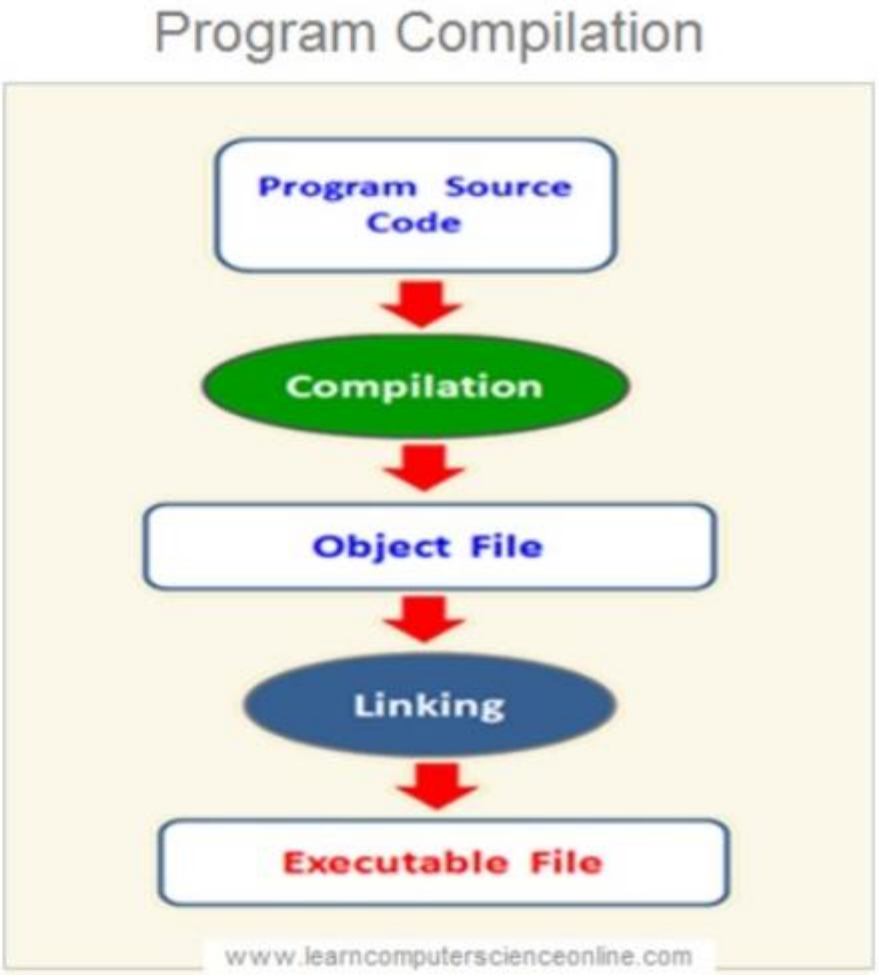
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

2.2 Machine Language



2.2 Machine Language

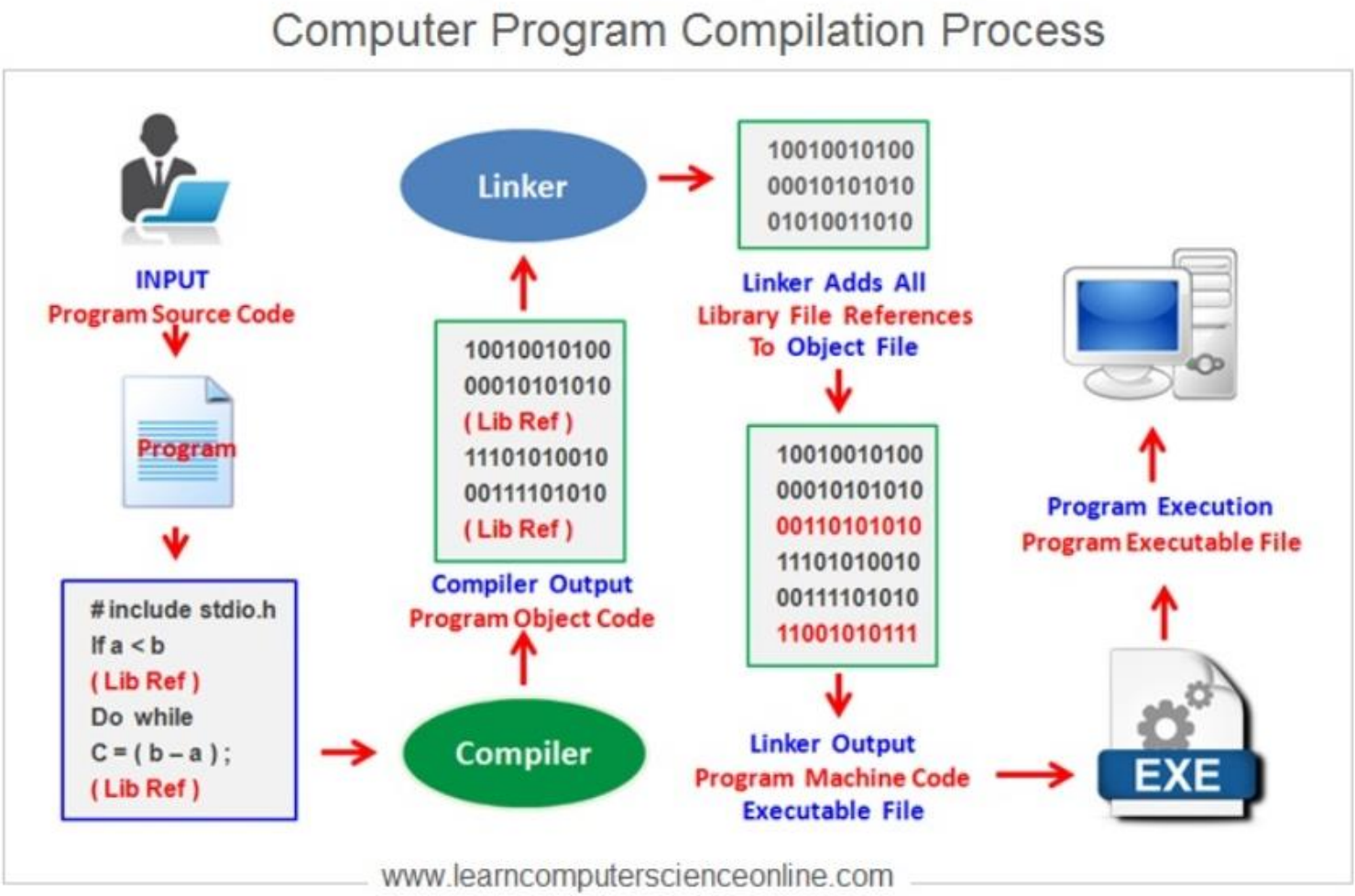




Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

2.2 Machine Language

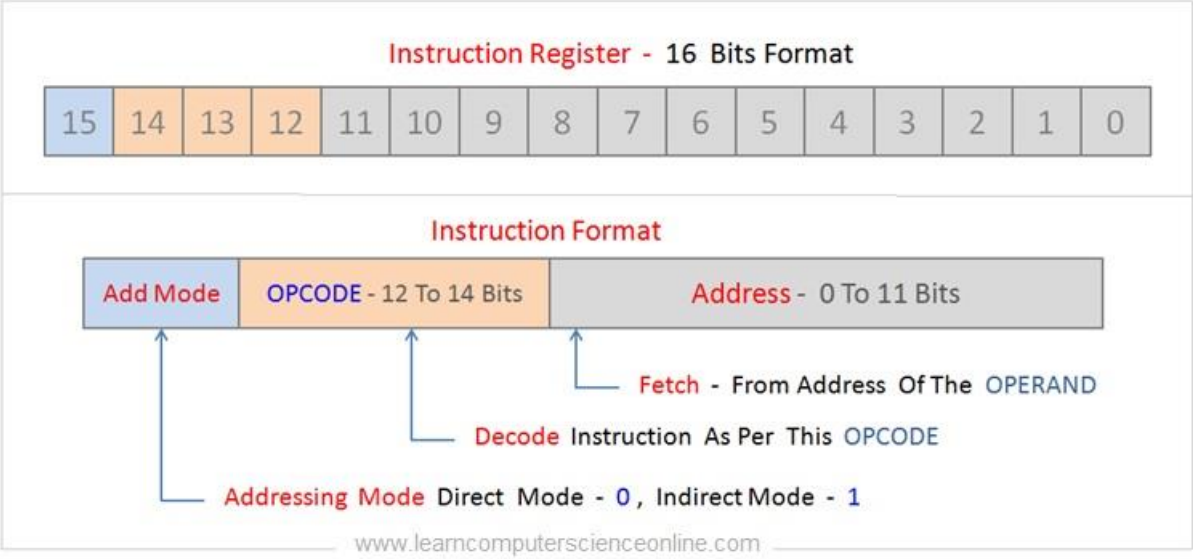


Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

2.2 Machine Language

- **Machine instruction:** An instruction encoded as a bit pattern recognizable by the CPU
- **Machine language:** The set of all instructions recognized by a machine

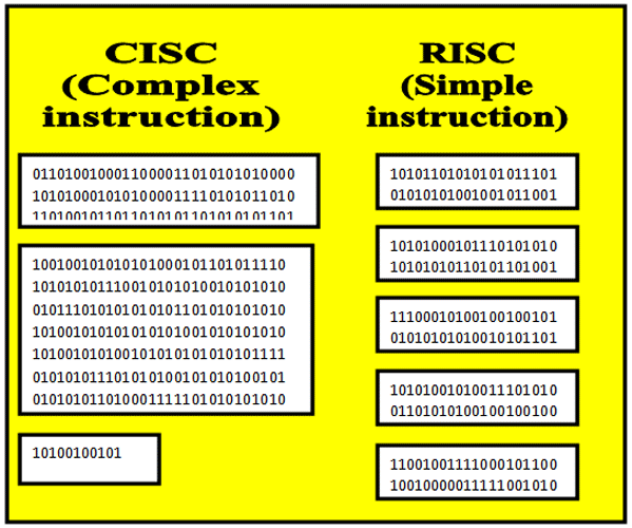


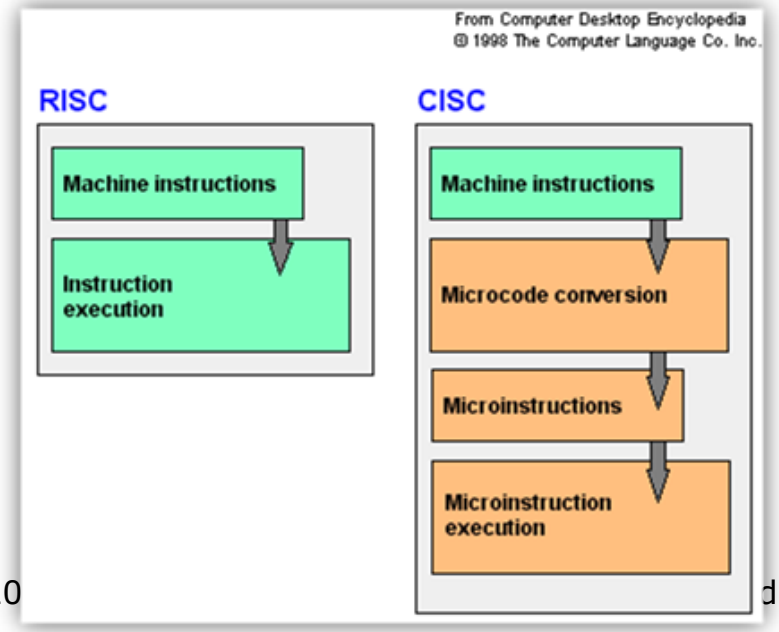
Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Machine Language Philosophies

- Reduced Instruction Set Computing (RISC)
  - Few, simple, efficient, and fast instructions
  - Examples: PowerPC from Apple/IBM/Motorola and ARM
- Complex Instruction Set Computing (CISC)
  - Many, convenient, and powerful instructions
  - Example: Intel





Pearson
Copyright © 20 d.

Machine Instruction Types

- Data Transfer: copy data from one location to another (e.g. LOAD, STORE)
- Arithmetic/Logic: operations on bit patterns (e.g. +, -, *, /, AND, OR, SHIFT, ROTATE)
- Control: direct the execution of the program (e.g. JUMP, BRANCH)

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Machine Instruction Types

Machine instruction formats.

Instructions using three registers	Opcode [31 – 26]	Operand_reg_1 [25 – 21]	Operand_ reg_2 [20 – 16]	Result_reg [15 – 11]	Unused [10 – 0]
Instructions using < 3 registers	Opcode [31 – 26]	Operand_reg_1 [25 – 21]	Result_reg [20 – 16]	Offset [15 – 0]	

ISA instruction definition table.

Mnemonic	Opcode	Definition
ADD	000001	Result_reg ← Operand_reg_1 + Operand_reg_2
LOAD	000010	Result_reg ← Data_memory[Operand_reg_1 + Offset]
STORE	000011	Data_memory[Operand_reg_1 + Offset] ← Operand_reg_2
JUMP	000100	Next_instr_ptr ← Operand_reg_1 + Offset
BRR	000101	Next_instr_ptr ← Default_next_instr_ptr + Offset
BEQ	000110	If Operand_reg_1 == Operand_reg_2 then Next_instr_ptr ← Default_next_instr_ptr + Offset
BNE	000111	If Operand_reg_1 != Operand_reg_2 then Next_instr_ptr ← Default_next_instr_ptr + Offset
LOADHI	001000	Result_reg[bits 31 – 16] ← Offset
SRL	001001	Result_reg ← Logical-right-shift of Operand_reg_1 by Offset bit positions
ADDI	001010	Result_reg ← Operand_reg_1 + Offset

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.2 Adding values stored in memory

Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.3 Dividing values stored in memory

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.4 The architecture of the Vole, as described in Appendix C

pendix C

Vole: A Simple Machine Language

In this appendix, we present a simple but representative machine language, named for the simple and hypothetical machine it would run on. We begin by explaining the architecture of the machine itself.

The Vole Architecture[1]

Let us say that our *Vole* computer has 16 general-purpose registers numbered 0x0 through 0xF. Each register is one byte (eight bits) long. For identifying registers within instructions, each register is assigned the unique four-bit pattern that represents its register number. Thus, register 0x0 is identified by binary 0000, and register 0x4 is identified by binary 0100.

There are 256 cells in the Vole's main memory. Each cell is assigned a unique address consisting of an integer in the range of 0 to 255. An address can therefore be represented by a pattern of eight bits ranging from 00000000 to 11111111 (or a hexadecimal value in the range of 0x00 to 0xFF).

Floating-point values are assumed to be stored in an eight-bit format discussed in Section 1.7 and summarized in Figure 1.24.

The Vole's Machine Language

Each Vole machine instruction is two bytes long. The first 4 bits provide the op-code; the last 12 bits make up the operand field. The table that follows lists the instructions in hexadecimal notation together with a short description of each. The letters R, S, and T are used in place of hexadecimal digits in those fields representing a register identifier that varies depending on the particular application of the instruction. The letters X and Y are used in lieu of hexadecimal digits in variable fields not representing a register.

[1] *Vole* refers to a family of stout, short-tailed rodent species found across much of the world. Like the processor described in this appendix, they are small but effective.

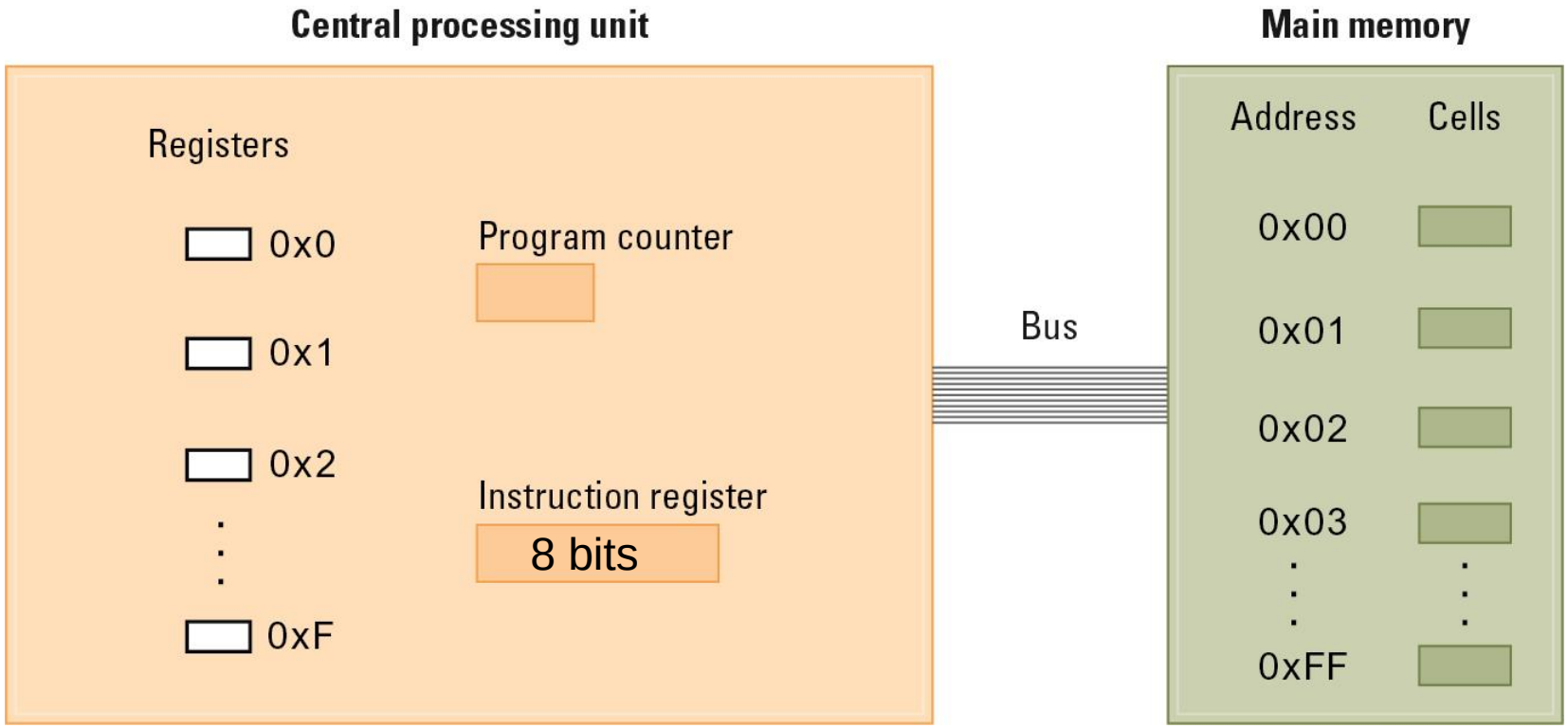
Op-code	Operand	Description
0x1	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY. *Example:* 0x14A3 would cause the contents of the memory cell located at address 0xA3 to be placed in register 0x4.
0x2	RXY	LOAD the register R with the bit pattern XY. *Example:* 0x20A3 would cause the value 0xA3 to be placed in register 0.
0x3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY. *Example:* 0x35B1 would cause the contents of register 0x5 to be placed in the memory cell whose address is 0xB1.
0x4	0RS	MOVE the bit pattern found in register R to register S. *Example:* 0x40A4 would cause the contents of register 0xA to be copied into register 0x4.
0x5	RST	ADD the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R. *Example:* 0x5726 would cause the binary values in registers 0x2 and 0x6 to be added and the sum placed in register 0x7.
0x6	RST	ADD the bit patterns in registers S and T as though they represented values in floating-point notation and leave the floating-point result in register R. *Example:* 0x634E would cause the values in registers 0x4 and 0xE to be added as floating-point values and the result to be placed in register 0x3.
0x7	RST	OR the bit patterns in registers S and T and place the result in register R. *Example:* 0x7CB4 would cause the result of ORing the contents of registers 0xB and 0x4 to be placed in register 0xC.
0x8	RST	AND the bit patterns in registers S and T and place the result in register R. *Example:* 0x8045 would cause the result of ANDing the contents of registers 0x4 and 0x5 to be placed in register 0x0.
0x9	RST	XOR the bit patterns in registers S and T and place the result in register R. *Example:* 0x95F3 would cause the result of XORing the contents of registers 0xF and 0x3 to be placed in register 0x5.
0xA	R0X	ROTATE the bit pattern in register R one bit to the right X times. Each time, place the bit that started at the low-order end at the high-order end. *Example:* 0xA403 would cause the contents of register 0x4 to be rotated 3 bits to the right in a circular fashion.

Op-code	Operand	Description
0xB	RXY	JUMP to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. (The jump is implemented by copying XY into the program counter during the execute phase.) *Example:* 0xB43C would first compare the contents of register 0x4 with the contents of register 0x0. If the two were equal, the pattern 0x3C would be placed in the program counter so that the next instruction executed would be the one located at that memory address. Otherwise, nothing would be done and program execution would continue in its normal sequence.
0xC	000	HALT execution. *Example:* 0xC000 would cause program execution to stop.

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.4 The architecture of the Vole, as described in Appendix C



Pearson
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

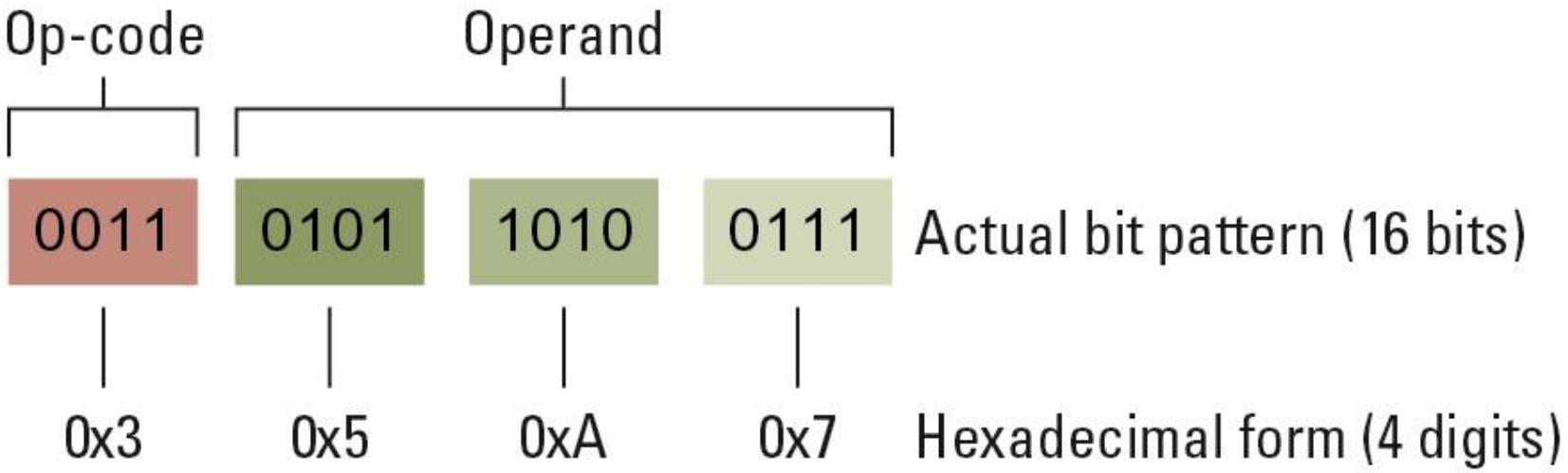
Parts of a Machine Instruction

- **Op-code:** Specifies which operation to execute
- **Operand:** Gives more detailed information about the operation
  - Interpretation of operand varies depending on op-code

Pearson

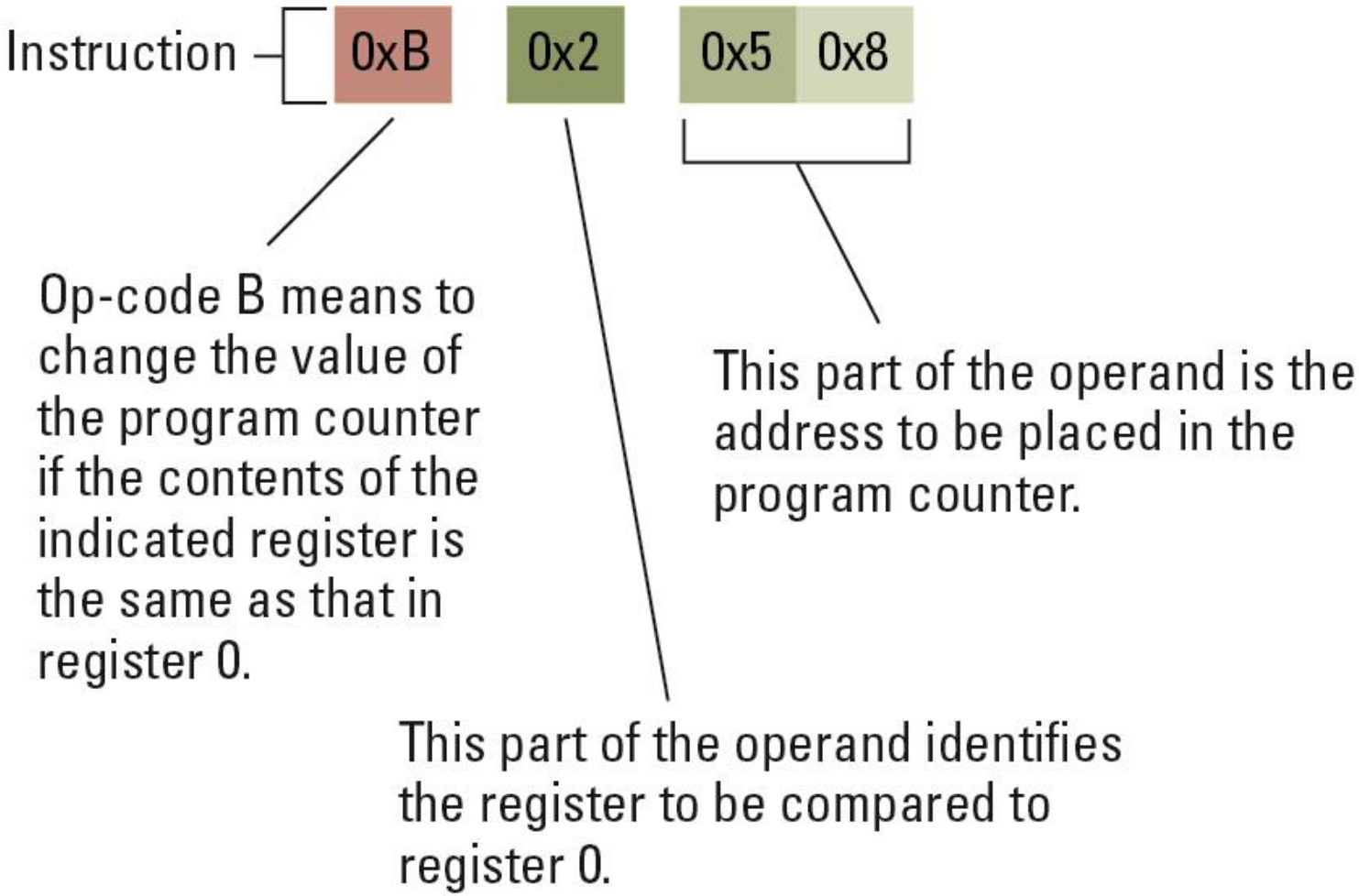
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.5 The composition of a Vole instruction



Pearson
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.6 Decoding the instruction 0x35A7



Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.7 An encoded version of the instructions in Figure 2.2

Encoded instructions	Translation
0x156C	Load register 0x5 with the bit pattern found in the memory cell at address 0x6C.
0x166D	Load register 0x6 with the bit pattern found in the memory cell at address 0x6D.
0x5056	Add the contents of register 0x5 and 0x6 as though they were two's complement representation and leave the result in register 0x0.
0x306E	Store the contents of register 0x0 in the memory cell at address 0x6E.
0xC000	Halt.

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

2.3 Program Execution

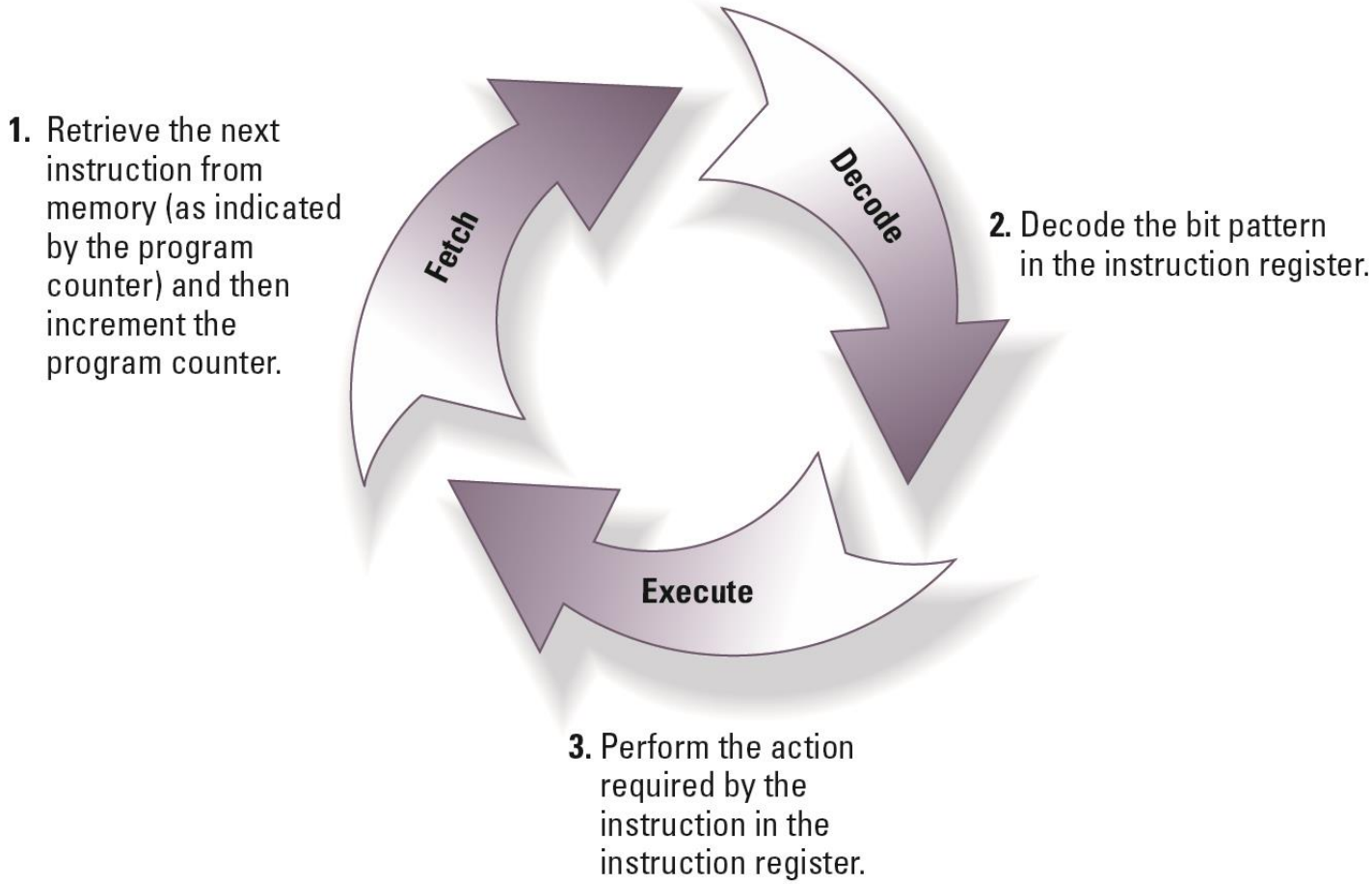
- Controlled by two special purpose registers
  - Instruction register
    - holds current instruction
  - Program counter
    - holds address of next instruction

- Machine Cycle: (repeat these 3 steps)
  - Fetch, Decode, Execute

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

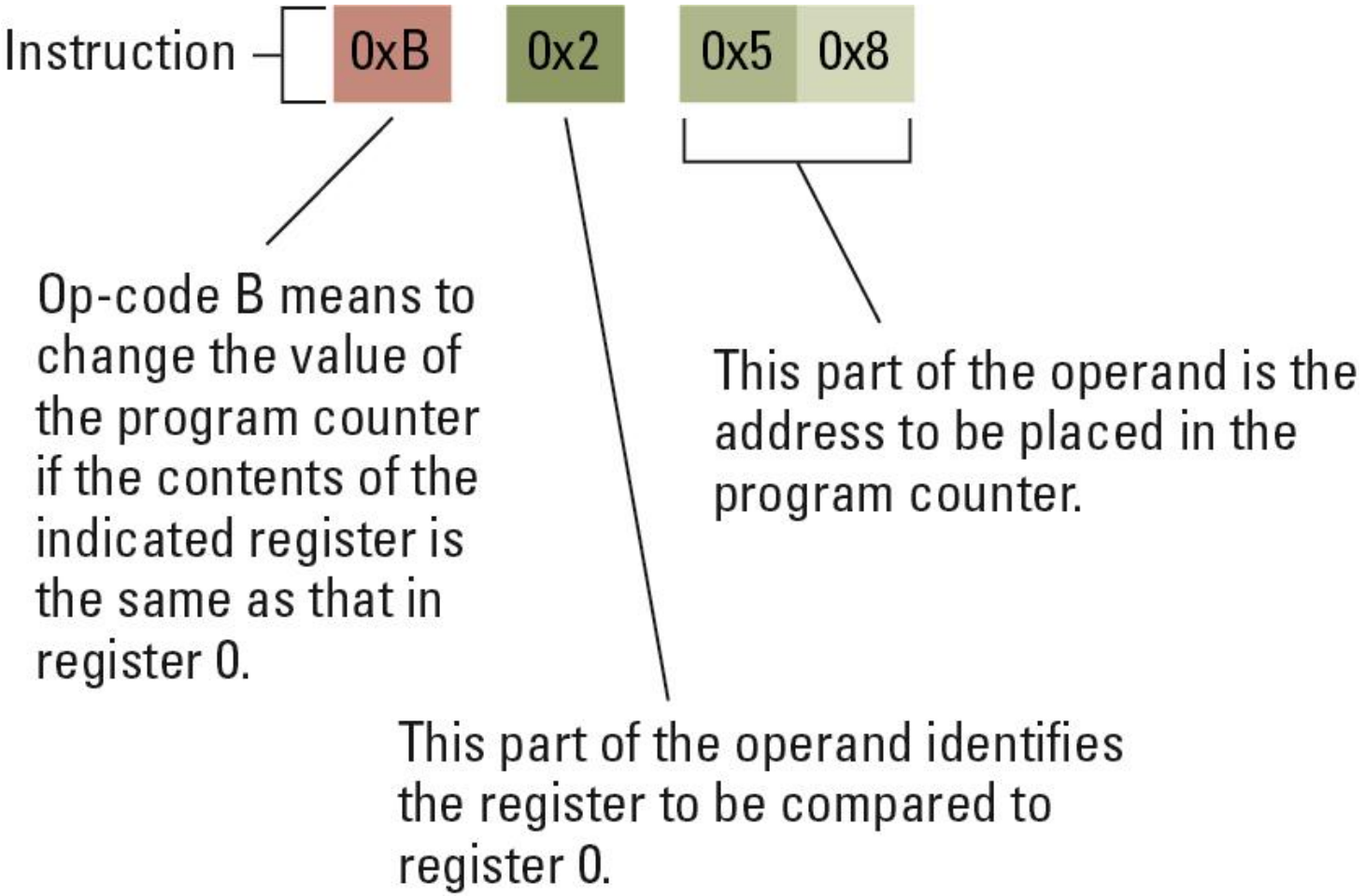
Figure 2.8 The machine cycle



Pearson

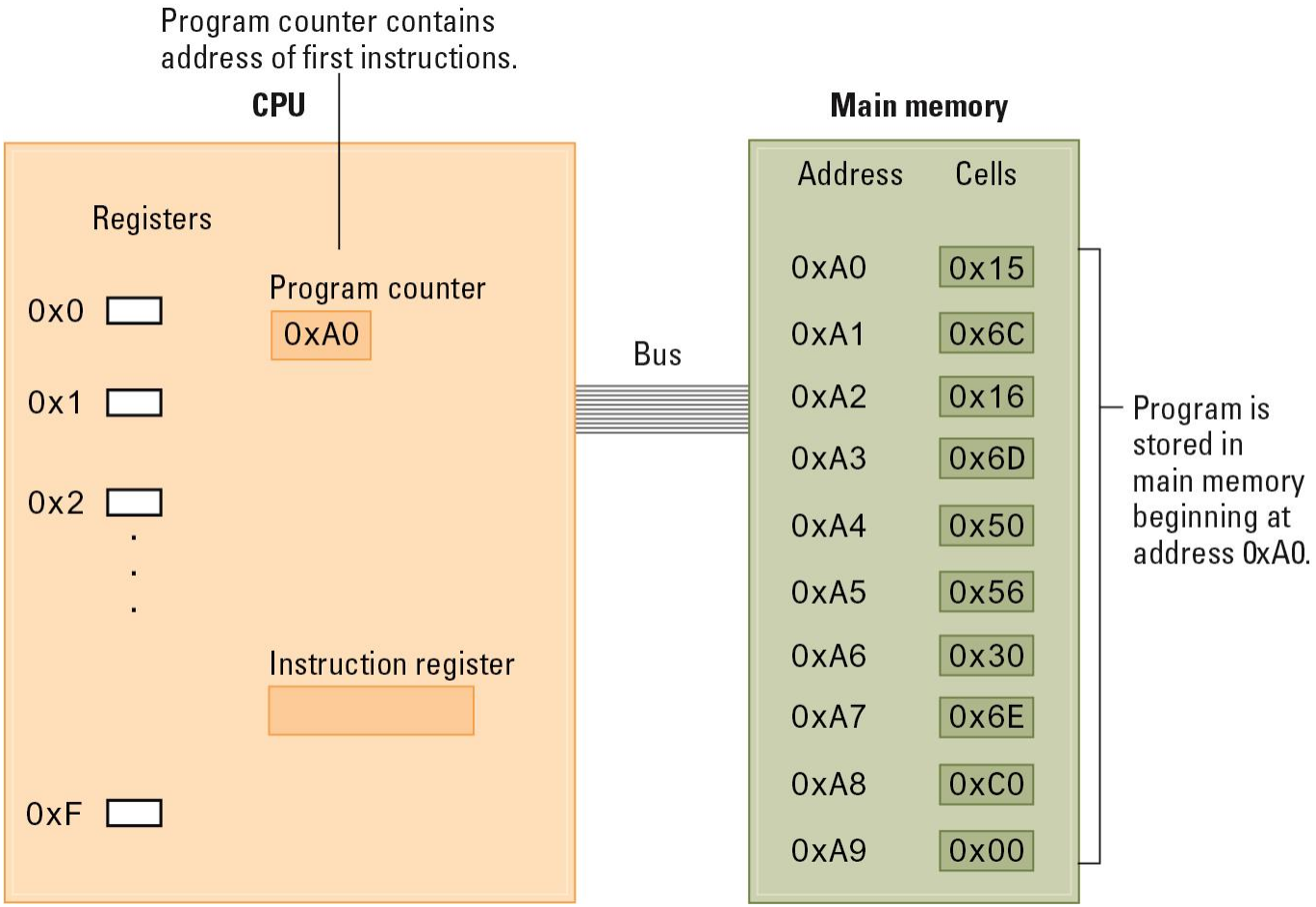
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.9 Decoding the instruction B258



Pearson
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

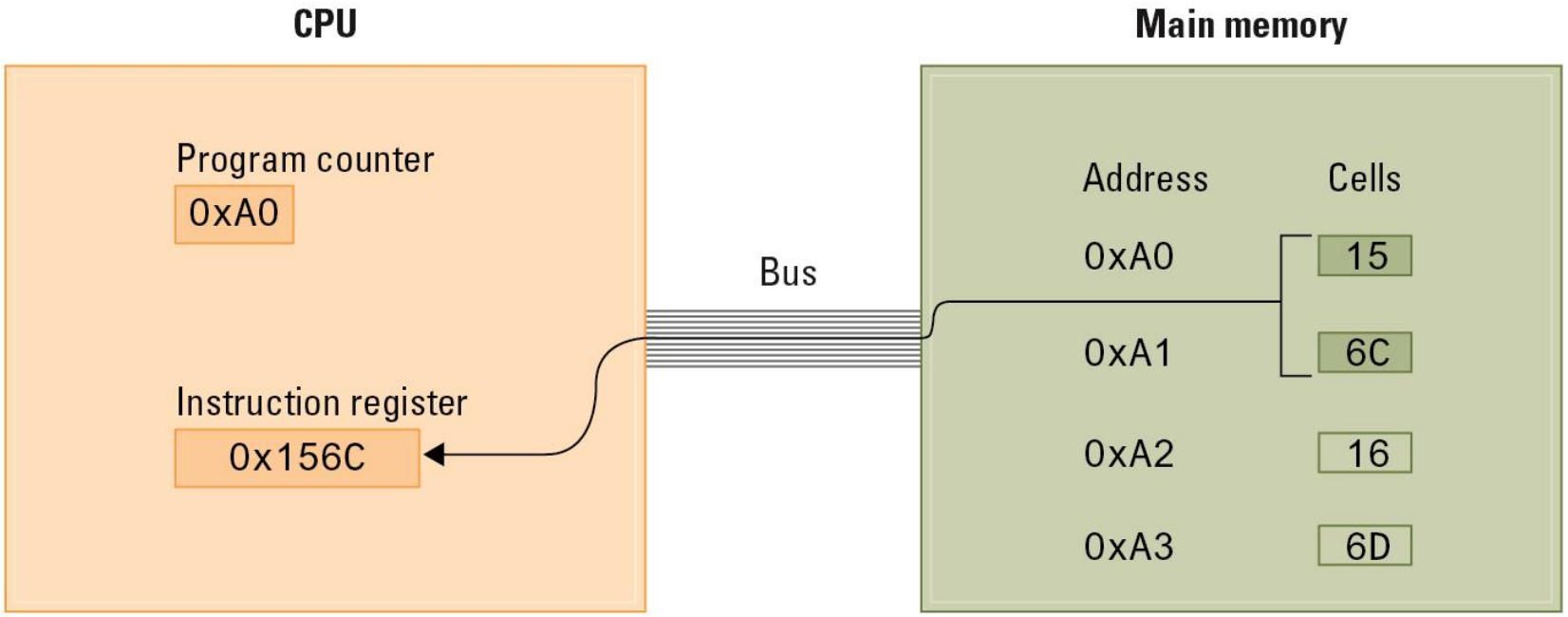
Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution



Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.11 Performing the fetch step of the machine cycle

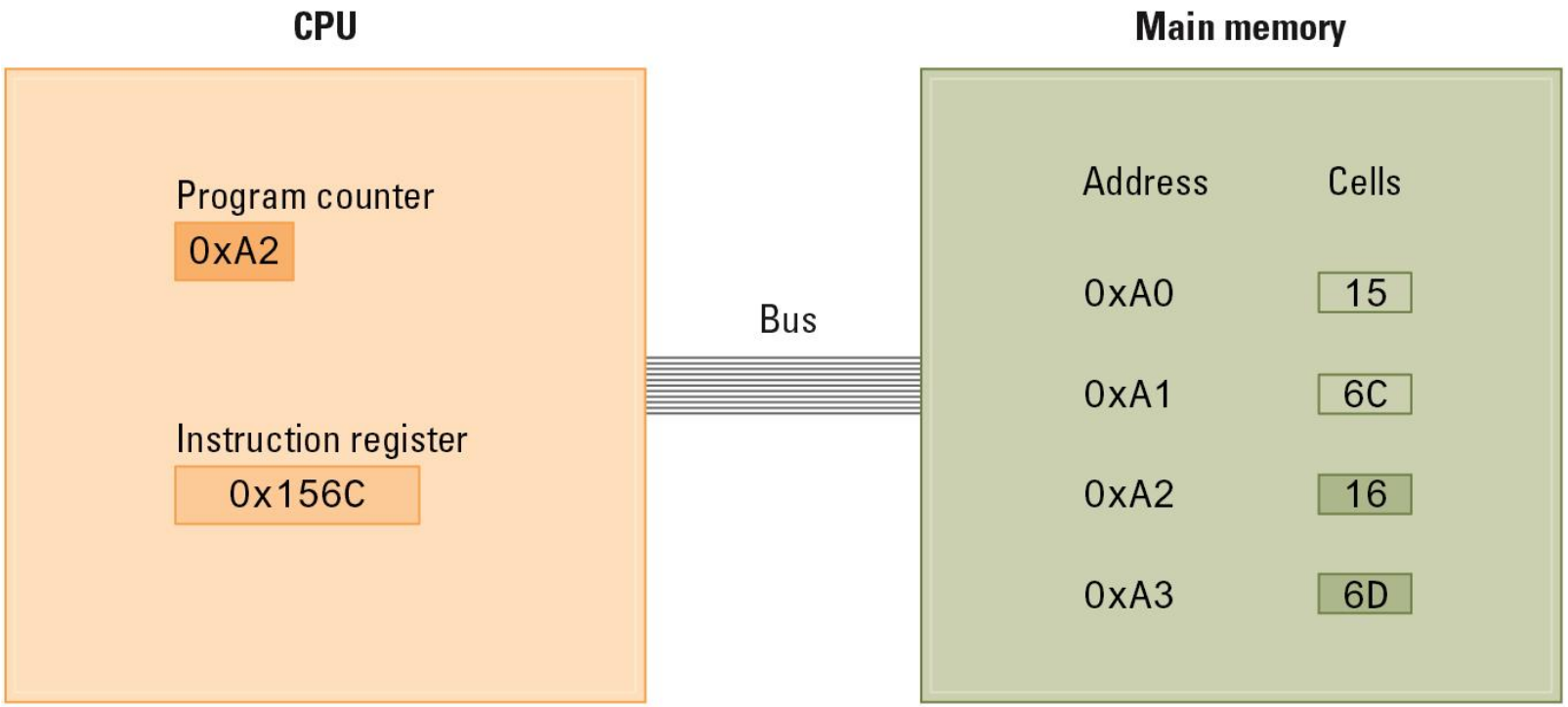


a. At the beginning of the fetch step, the instruction starting at address 0xA0 is retrieved from memory and placed in the instruction register.

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.11 Performing the fetch step of the machine cycle (continued)



b. Then the program counter is incremented so that it points to the next instruction.

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

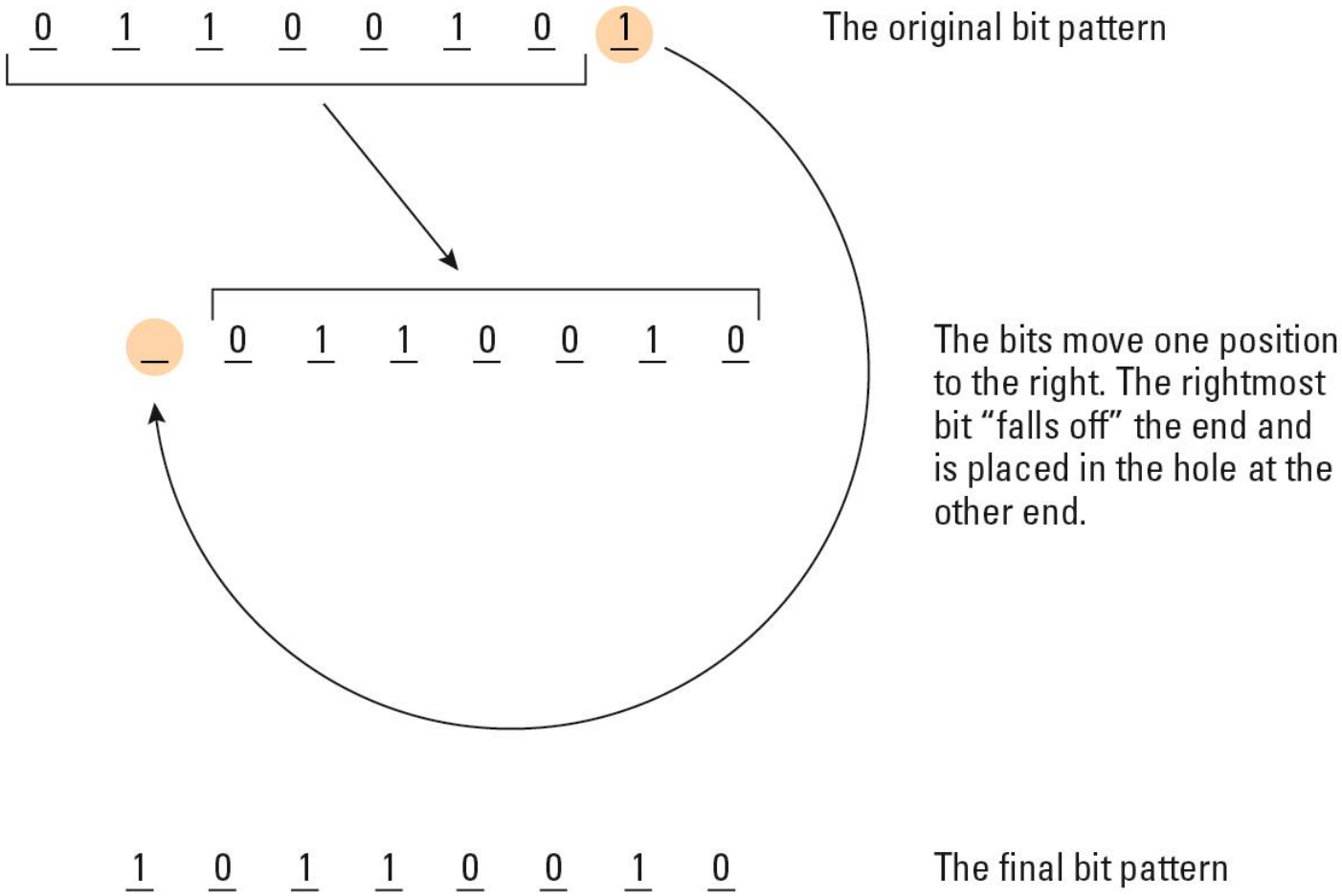
2.4 Arithmetic/Logic Instructions

- Logic Operations:
  - AND, OR, XOR
  - often used to mask an operand
- Rotation and Shift Operations:
  - circular shift, logical shift, arithmetic shift
- Arithmetic Operations:
  - add, subtract, multiply, divide
  - two’s complement versus floating-point

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.12 Rotating the bit pattern 0x65 one bit to the right



Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

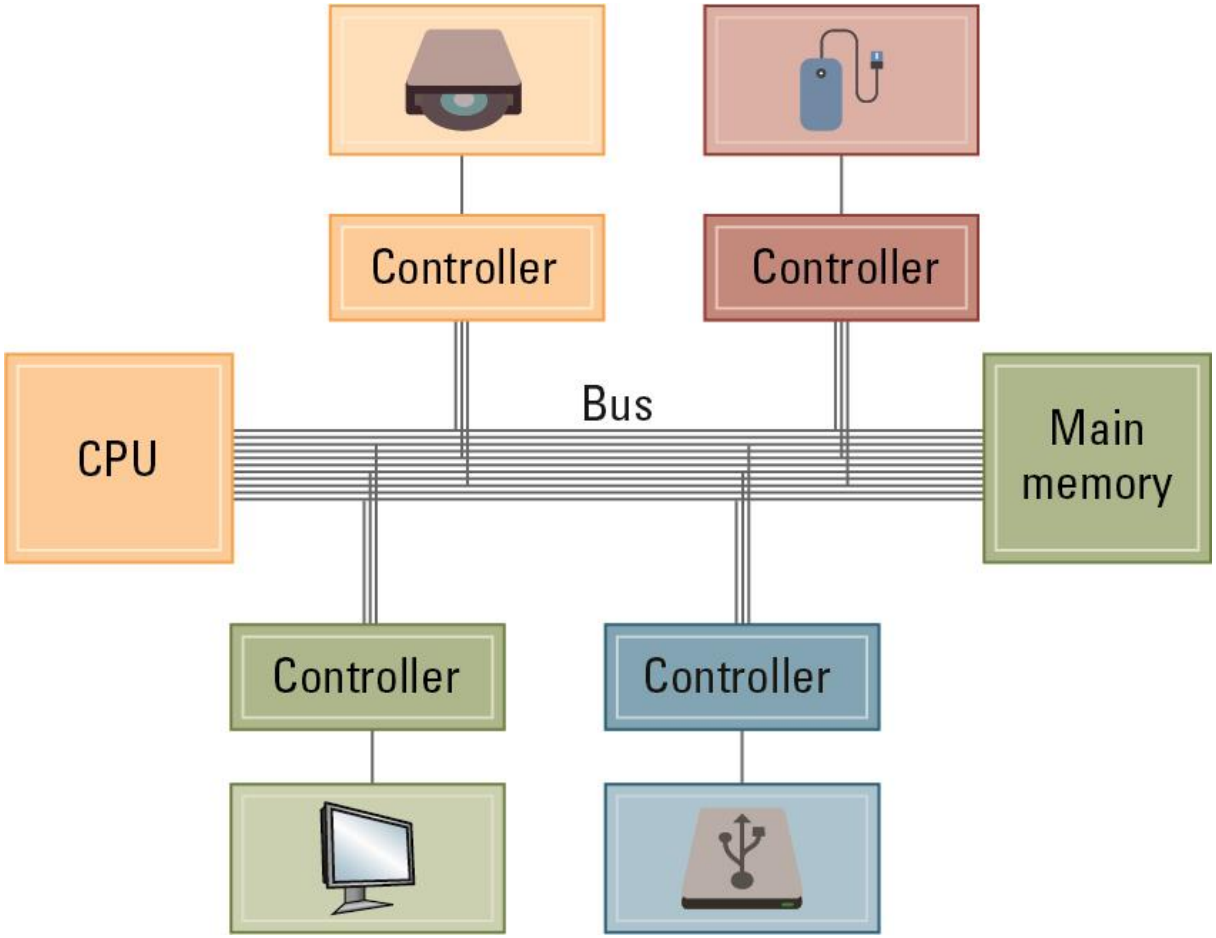
2.5 Communicating with Other Devices

- **Controller:** handles communication between the computer and other devices
  - Specialized (by type of device)
  - General purpose (USB, HDMI)
- **Port:** The point at which a device connects to a computer
- **Memory-mapped I/O:** devices appear to the CPU as though they were memory locations

Pearson

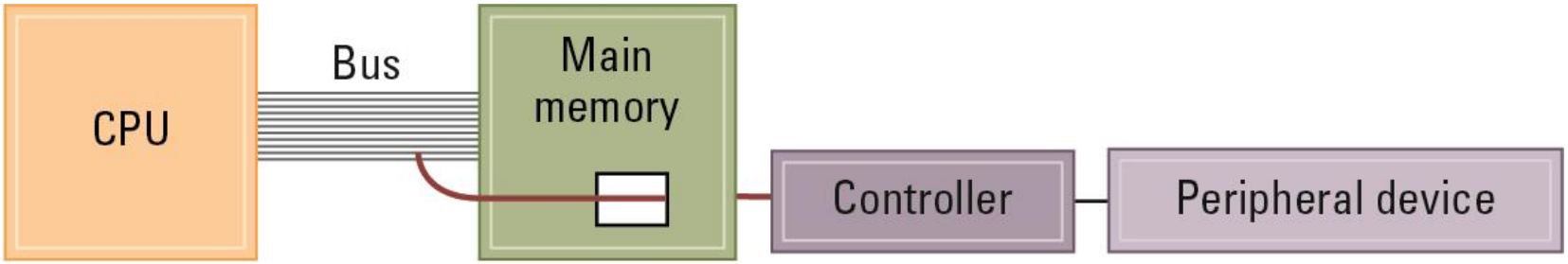
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.13 Controllers attached to a machine's bus



Pearson
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.14 A conceptual representation of memory-mapped I/O



Pearson
Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Communicating with Other Devices (continued)

- **Direct memory access (DMA):** Main memory access by a controller over the bus
  - **Von Neumann Bottleneck:** occurs when the CPU and controllers compete for bus access
- **Handshaking:** the process of coordinating the transfer of data between the computer and the peripheral device

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Communicating with Other Devices (continued)

- **Popular Communication Media**
  - **Parallel Communication:** Several signals transferred at the same time, each on a separate "line" (computer's internal bus)
  - **Serial Communication:** Signals are transferred one after the other over a single "line" (USB, FireWire)

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Data Communication Rates

- Measurement units
  - bps: bits per second
  - Kbps: Kilo-bps (1,000 bps)
  - Mbps: Mega-bps (1,000,000 bps)
  - Gbps: Giga-bps (1,000,000,000 bps)
- Bandwidth: Maximum available rate

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

2.6 Programming Data Manipulation

- Programing languages shields users from details of the machine:
  - A single Python statement might map to one, tens, or hundreds of machine instructions
  - Programmer does not need to know if the processor is RISC or CISC
  - Assigning variables surely involves LOAD, STORE, and MOVE op-codes

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Bitwise Problems as Python Code

```
print(bin(0b10011010 & 0b11001001))
# Prints '0b10001000'

print(bin(0b10011010 | 0b11001001))
# Prints '0b11011011'

print(bin(0b10011010 ^ 0b11001001))
# Prints '0b1010011'
```

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Control Structures

- If statement:

```
if (water_temp > 140):
    print('Bath water too hot!')
```

- While statement:

```
while (n < 10):
    print(n)
    n = n + 1
```

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Functions

- **Function**: A name for a series of operations that should be performed on the given parameter or parameters
- **Function call**: Appearance of a function in an expression or statement

```
x = 1034
y = 1056
z = 2078
biggest = max(x, y, z)
print(biggest)  # Prints '2078'
```

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Functions (continued)

- **Argument Value**: A value plugged into a parameter
- **Fruitful** functions **return** a value
- **void functions**, or **procedures,** do not return a value

```
sideA = 3.0
sideB = 4.0
# Calculate third side via Pythagorean Theorem
hypotenuse = math.sqrt(sideA**2 + sideB**2)
print(hypotenuse)
```

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Input / Output

```
# Calculates the hypotenuse of a right triangle

import math

# Inputting the side lengths, first try
sideA = int(input('Length of side A? '))
sideB = int(input('Length of side B? '))

# Calculate third side via Pythagorean Theorem
hypotenuse = math.sqrt(sideA**2 + sideB**2)

print(hypotenuse)
```

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Marathon Training Assistant

```
# Marathon training assistant.

import math

# This function converts a number of minutes and
# seconds into just seconds.
def total_seconds(min, sec):
    return min * 60 + sec

# This function calculates a speed in miles per hour
given
# a time (in seconds) to run a single mile.
def speed(time):
    return 3600 / time
```

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Marathon Training Assistant (continued)

```
# Prompt user for pace and mileage.
pace_minutes = int(input('Minutes per mile? '))
pace_seconds = int(input('Seconds per mile? '))
miles = int(input('Total miles? '))

# Calculate and print speed.
mph = speed(total_seconds(pace_minutes, pace_seconds))
print('Your speed is ' + str(mph) + ' mph')

# Calculate elapsed time for planned workout.
total = miles * total_seconds(pace_minutes, pace_seconds)
elapsed_minutes = total // 60
elapsed_seconds = total % 60

print('Your elapsed time is ' + str(elapsed_minutes) +
              ' mins ' + str(elapsed_seconds) + ' secs')
```

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

Figure 2.15 Example Marathon Training Data

Time Per Mile				Total Elapsed Time	
Minutes	Seconds	Miles	Speed (mph)	Minutes	Seconds
9	14	5	6.49819494584	46	10
8	0	3	7.5	24	0
7	45	6	7.74193548387	46	30
7	25	1	8.08988764044	7	25

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.

2.7 Other Architectures

- Technologies to increase throughput:
  - Pipelining: Overlap steps of the machine cycle
  - Parallel Processing: Use multiple processors simultaneously
    - SISD: Single Instruction, Single Data
      - No parallel processing
    - MIMD: Multiple Instruction, Multiple Data
      - Different programs, different data
    - SIMD: Single Instruction, Multiple Data
      - Same program, different data

Pearson

Copyright © 2019 Pearson Education, Ltd. All Rights Reserved.